Computation/AIS

# Software Engineering Newsletter

Sept-Oct 95

*Published by the Software Engineering Working Group and the Software Technology Center to promote software engineering education*

*UCRL-AR-121011-95-*

## Events in Review

**August 15**

There was a good turnout in attendance for *GRADY BOOCH*, from the Rational Software Corporation, who talked about what's up with ***Object-Oriented Methods***. See Article on Page 2.

**August 18**

A seminar was held by Steve Wong, Terri Quinn, and Al Leibee to talk about some of the problems in setting up a ***Metrics*** program, and presented a possible approach for putting one together.

**September 18**

Don Shilling and Don Rathbun from Allied Signal Aerospace, Kansas City Division, spoke about ***ISO 9000*** and their experiences becoming certified. Please see the enclosed article on ISO 9000, by Carolyn Owens. For viewgraphs call the STC office at X3-8543 or e-mail to stc.llnl.gov.

*Please refer to Al Leibee X2-1665 or Jennifer Gibson X3-8543 for questions regarding any STC functions.*

**STC WWW Pages**

*External: http://www.llnl.gov/stc/stc.html*

*or*

*Internal: http://www.llnl.gov/llnl_only/stc/*

## Upcoming Events

**October 9**

Rick Ball, from Richard Ball and Associates in Canada, will hold a Maintenance Seminar for 9-12 in Bldg. 439 Training Room.

***October 30 and 31***

Roger Bate From SEI in Pittsburgh will be coming out to give a seminar on Systems Engineering Capability Maturity Model and Integrated Product Development Capability Maturity Model.

**November 9**

STC is planning a Testing Tools expo, hosted by Federal Business Council will be hosting a Vendor Demonstration Day, in Bldg. 132 white area.

**Inside this issue:**

## Grady Booch speaks about software development methods

*Jeff Young,*
*X3-8333, jeffyoung@llnl.gov, L-548*

If anyone was destined to become well know in the Object Orientation (OO) field, it was Grady Booch. Booch, whose name is almost synomonous with object orientation, spoke to a receptive Lab audience in August. The formal title of Booch's talk was "The Maturation of Object-Oriented Methods," but the lecture itself was far from formal. Booch sat atop a table as he regailed his audience with OO tales.

### OO and C++: no silver bullets

To no one's surprise, Booch stated that the dominate OO language is C++. However, he pointed out that he is relieved that the new worldwide air traffic control software is written in Ada, not C++. OO is not tied to a specific language, but rather certain languages facilitate OO development and all languages have their downside. Writing in C++ does not make you object oriented, Booch declared.

**"Someone in my family had the foresight to spell our name with two o's ...." Grady Booch**

Booch cited an example of a shop that had programed in C++ for years, and they bragged about it. Then, when they were looking at Rational's tools, they asked that their code be analyzed. As they handed the diskette over, they asked "Do there have to be any classes for the analysis to work?" In another example, Booch discussed how a similar shop had exactly one method per class, i.e., something akin to "do it".

Booch has seen large projects succeed and fail with OO. The largest successful OO project, he reports, had nearly 10,000 classes. However, the success or failure rate tends to depend upon aspects other than the analysis and design method. For example, Booch recalled a failure that probably had more to do with the 8,000-page-requirements specification than with any design method. Some of the characteristics he has observed in failed projects include: requirements that are out of control, software with no sense of architecture, and computers with configuration management problems.

There are many problems that a project can encounter that can put the project at risk. However, there are many companies that have enough resources to overcome problems by brute force if by nothing else. Booch calls this the "Microsoft effect" and defines it as "a company can reach a certain critical mass beyond which they can buy their way out of a problem."

For those projects that decide to use an OO approach, there are many OO methods to choose from. These methods include Booch, OMT, Objectory, Schlaer-Mellor, and Coad/Yourdon, just to mention a few. By far, the dominate software development method is called Chaos (heroic programming). Chaos, however, is not a sustainable business practice. Methods are important because "methods help us mitigate the Microsoft effect."

### The new method

While there are many methods, these methods have been converging with similar concepts and processes. Now, Booch and Rumbaugh (a principal author of the OMT method) have decided to go in the same direction Rather than pursuing the same concepts independently, and coming up with similar but still different methods, they are merging and to gain the benefits of a unified notation.

The new notation will use rectangles for classes and structured clouds for objects; it will be unveiled (version 0.8) at OOPSLA (spell out) this month. The unified methodology will have the visual abstractions necessary to model distributed and concurrent systems. Next spring is the target date for releasing version 1.0, at which time they plan to seek standardization.

New systems tend to be large distributed systems. Concurrency is also coming into play: new systems are moving away from the mainframe and PC environments to environments where the "network is the computer". Booch and Rumbaugh are working on the visual abstractions necessary to model distributed and concurrent systems.

*Continued on Page 3*

*Continued from page 2*

## Grady Booch speaks about software development methods

**Design patterns**

Booch asked the audience "Who is developing OO systems?" Surprisingly, only a small percentage of the audience responded. (Are we shy or is this an indication that OO designs are not commonplace at LLNL?). Booch then started an OO bidding war by asking who has the system with the most classes. The responses trickled in...10...40...500! The winning respondent was Joann Matone of the Conflict Simulation Lab. He then asked Joann what are the top lessons they learned:

- NEVER trust your first class
- Keep your code out of your class definitions

Booch agreed with both of these. Then he described the 100-200 class boundary where classes do not seem to be sufficient (though they are still necessary). Booch says that for large well structured OO programs the behavior of the system transcends classes. This transcending behavior shows up in class clusters or patterns. There are three categories of patterns: idioms, mechanisms, and frameworks. Idioms are patterns close to the language, like a dialect. Idioms are usually handled in style guides. Mechanisms are a collaboration of architectures. His example here was a blackboard system where knowledge sources use their growing knowledge on the blackboard to add their solution. Finally, frameworks are for projects that will be decades long. Frameworks are micro architectures made up of a set of classes and architectures. He gave an example of a company that developed their frameworks up front and now they get 60-70% reuse (without modifications) which allows them to be responsive to requirements changes, thereby becoming a leader their market.

**Scenarios**

Another powerful technique of object oriented analysis and design are scenarios or use cases as described by Jacobsen. Use cases allow you to articulate scenarios that wind their way through the system. Use cases are ways to use the system as expressed by the users, and, therefore, provide a means to communicate with real users of the system. The output of a use case analysis is a set of scripts as to how the system will be used. In each use case you will find multiple scenarios: primary scenarios and secondary scenarios.

According to Booch, it's easy to find about 70% of your objects in analysis with use cases: you pick the important use cases and their primary scenarios. The next 25% of your objects are found in the architectural design phase. The last 5%, he said, are found in maintenance. The discovery of the final few classes in maintenance can have a positive effect on the overall system.

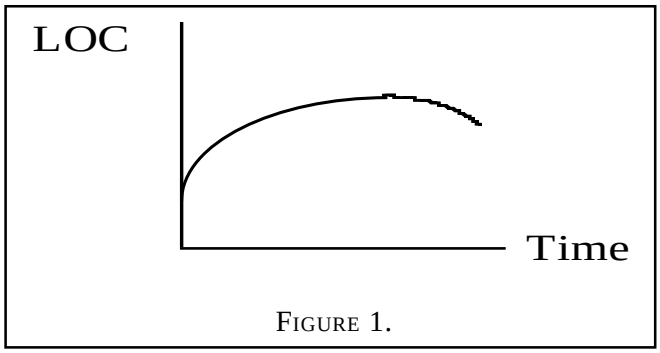


FIGURE 1.

By adding additional classes it allows you to see the patterns and bring the application together in a way that simplifies the design. Hyper-productive organizations achieve this effect and actually reduce the size of their code (see figure 1). A counter example is Microsoft, where they build their software almost every night. In this case, if a developers changes break the build they are under pressure to correct it before the next build. That kind of environment is not conducive to making simplifying changes. In response to a question from the audience, Booch indicated that Rational builds their software about once a week.

**Questions From The Audience**

- What metrics are there to show a greater or lessor object design? Booch has his own metrics.
- How many classes in the system. He has a heuristic about how many classes per day a

*More questions from the audience answered on page 4*

# Why Isn't Cleanroom the Universal Software Development Methodology? (Part Two)

*Reprinted from* CrossTalk *vol 8 No. 5, Reprinted with Permission*
*Johnnie Henderson, Software Technology Support Center*
*801-777-8057 DSN 777-8057*
*Internet: hendersj@software.hill.af.mil*

*Editor's note: We have broken this article into two parts. Part two follows; part one was published in the July - Aug. issue of this newsletter. We kept all the content but we rearranged it for publication. Part two reprints the introduction and adds the section "Unit Testing Vs. Correctness Verification and Statistical Quality Control" which was in the middle of the original article.*

## Introduction

Cleanroom—a methodology that promises much lower error rates, higher productivity, and delivery of software on schedule and within budget—sounds like the proverbial "silver bullet" that the industry is looking for. Why hasn't it caught on and spread like wildfire? There are three basic reasons:

1. A belief that the Cleanroom methodology is too theoretical, too mathematical, and too radical for use in real software development.
2. It advocates no unit testing by developers but instead replaces it with correctness verification and statistical quality control—concepts that represent a major departure from the way most software is developed today.
3. The maturity of the software development industry. The use of Cleanroom processes requires rigorous application of defined processes in all lifecycle phases. Since most of the industry is still operating at the ad hoc level (as defined by the Software Engineering Institute Capability Maturity Model), the industry has not been ready to apply those techniques.

What does the experience in using Cleanroom say about whether or not these are valid concerns? This article attempts to answer that question.

## Unit Testing Vs. Correctness Verification and Statistical Quality Control

The fundamental approach to verification as espoused by Cleanroom is aimed at introducing mathematical reasoning, not mathematical notation into the verification process. The principal motivation is to provide a rigorous methodology for software development and to provide a firm foundation as an engineering discipline. Mathematical verification of programs is done by using a few basic control structures and defining proofs following rules specified in a correctness theorem. The proof strategy is divided into small parts that easily accumulate into proof for a large software system [4].

The method of human mathematical verification used in Cleanroom is called functional verification. Functional verification is organized around correctness proofs, which are defined for the design constructs used in a software design. Using this type of functional verification, the verification problem changes from one with an infinite number of combinations to consider to a finite process because the correctness theorem defines the required number of conditions that must be verified for each design construct used. It reduces software verification to ordinary mathematical reasoning about sets and functions [5]. The objective is to develop designs in concert with associated correctness proofs. Designs are created with the objective of being easy to verify. A rule of thumb followed is that when designs become difficult to verify they should be redone for simplicity [1,2].

Statistical quality control is used when you have too many items to test all of them exhaustively. Instead, you statistically sample and analyze some items and scientifically assess the quality of all of the items through extrapolation. This technique is widely used in manufacturing in which items in a production line are sampled, the quality is measured, then sample quality is extrapolated to the entire

*Continued on page 8..*

## Focus on Metrics: Measurement and Software Management

*Al Leibee, Leibee1@llnl.gov,*
*Ext. 2-1665, L-307*

The Software Engineering Laboratory (SEL) at NASA's Goddard Space Flight Center has been studying and applying various software measurement techniques since 1976. Over this period of time, the SEL itself has collected and analyzed measurement data from more 100 flight dynamics projects ranging in size from 10,000 to 1,000,000 source lines of code. This data has been used to generate models and relationships that are used in managing software development. The SEL has also produced a *Software Measurement Guidebook* (SEL-94-002, July 1994) which is aimed at helping projects begin or improve a measurement program. In this article, I summarize from the Guidebook the relationships between measurement and software management.

An understanding of a project's processes- what it does and how it does it, is required for planning, managing, and improving those processes. A quantified understanding through measurement of a project's software processes and products lets the project derive models of those processes. The project can then use these models to plan, manage, and improve its software development. Quantified understanding might include knowing:

- The size of the products.
- The effort spent in the life cycle activities.
- How resources are allocated and used throughout the life cycle.
- What types of errors and enhancements are typically made.
- How many defects are corrected.

The derived models can then be used to address day-to-day questions such as:

- How long will it take to complete testing?
- Can the schedule be compressed by adding staff?
- Is reliability a function of testing time?

The following table shows some possible areas of understanding and their associated measurements.

*Continued on page 6*

*Continued from page 3*

## Grady Booch speaks about software development methods

person can develop.

- How the classes are clustered
- The size and shape of the inheritance tree. His guidelines are 5±2 deep and 7±2 wide.
- Complexity (McCabe). He wants to see a bell curve; look at the bumps
- Stability. Groups of classes changing together is good. If changes are ad hoc across unrelated classes its a symptom of design rot.
- Can you go too far with OO? Yes. You can over architect in which case you tend NOT to do use cases. Your classes are too big. Need to get a tiger team to seek out patterns and expose them.
- When should you start your Graphical User Interface (GUI) design? The relative risk to the system due to the GUI determine when you start. If your application is heavily dependent on the GUI then you must start right away. He recommends, in general, that you start your GUI when you are doing your analysis (in the first month) and that you keep your architecture separate from your GUI. The GUI is a projection on the domain model. Further, he recommends using a human factors person. Visual Basic and Smalltalk could be used to build prototypes so you can expose your users early and get social buy-in.
- OO Databases and Relational distributed databases, what's the best to use? If you have a domain model with tight semantic coupling across objects then use an OODBMS. Otherwise, use an RDBMS with a thin layer on top.

*Continued from page 5*

## Focus on Metrics: Measurement and Software Management

| Understanding Area | Associated Measurements |
|---|---|
| The cost (resource) characteristics of software | Distribution of effort over development activities<br>Typical cost per line of code<br>Cost of maintenance<br>Hours spent on documentation<br>Computer resources required<br><br>Amount of rework expected<br>Number and classes of errors found during development or maintenance<br><br>How and when defects are found |
| The defect characteristics of software | Number and classes of defects found in specifications<br><br>Pass/fail rates for integration and system testing<br>Typical rate of growth of source code during development |
| The rate of source code production | Typical rate of change of source code during development or maintenance<br>Total number of lines of code produced |
| Relationship between amount of software to be developed and the duration of the project and the effort expended<br><br>The relationship between estimated software size and other key attributes | Schedule as a function of software size<br>Cost as a function of size<br>Total number of pages of documentation produced<br>Average staff size |

Many NASA programs have applied this strategy of using measurement to establish a quantified understanding of processes and products to derive models. One NASA organization collected and analyzed data on the distribution of 200 staff-years effort over development activities on 25 projects to build the following effort distribution model:

| | |
|---|---|
| Design Activities: | 23% of total effort |
| Coding Activities: | 21% of total effort |
| Testing Activities: | 30% of total effort |
| Other activities: (training, meetings, travel) | 26% of total effort |

Managers can use this model to estimate the effort required for a particular phase. For example, if the actual amount of time spent in Design and Coding were 200 staff-hours, then Testing would be estimated at 200 * 30 /(23 + 21) = 136 staff-hours.

*Continued on page 7*

*Continued from page 6*

## Focus on Metrics: Measurement and Software Management

Note that maintenance effort is not included in this model.

NASA also developed the following defect distribution model by collecting and classifying 10,000 errors over a period of 5 years from a large sample of NASA projects:

| | |
|---|---|
| Logic/Control Defects: | 16% |
| Computation Defects: | 15% |
| Data Defects: | 30% |
| Initialization Defects: | 15% |
| Interface Defects: | 24% |

With an environment-specific defect distribution model, defect-finding activities such as Walkthroughs can be focused on specific types.

NASA also developed a model of the typical growth of source code during development by collecting the rate at which source code was added to a project's controlled library for more than 20 projects. These projects followed the waterfall development life cycle. Figure 1 depicts the typical growth rate:

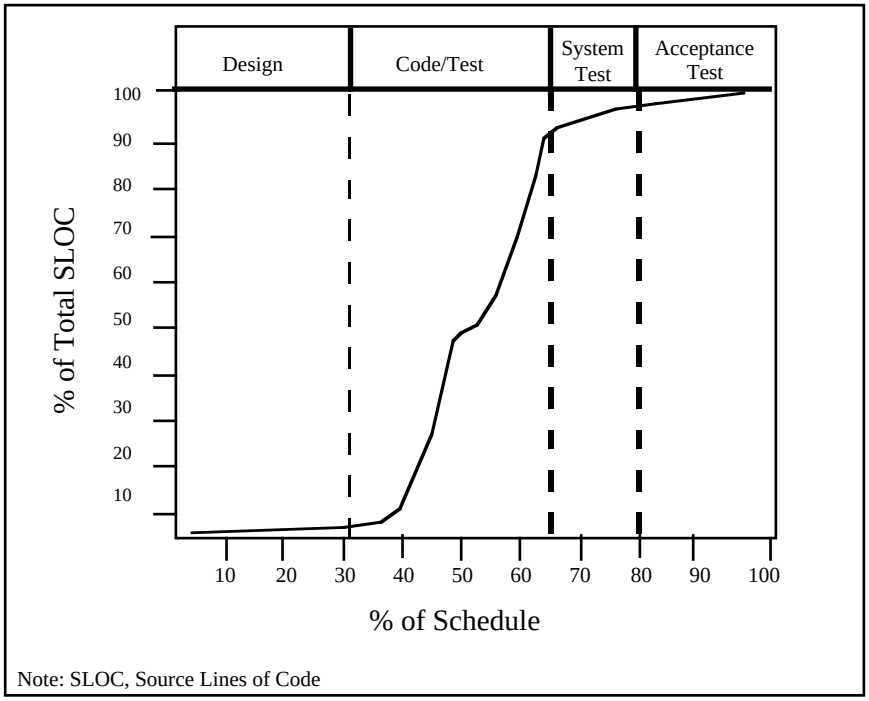


FIGURE 1: GROWTH RATE OF SOURCE

This model can be used by new projects to see if the project is progressing as expected or if the schedule is reasonable.

NASA also collected data to derive the following relationships between product parameters such as KSLOC (thousand source lines of code) and process parameters such as effort:

| | |
|---|---|
| Effort (in staff-months) | $=1.48 *(KSLOC)^{0.98}$ |
| Duration (in months) | $= 4.6 * (KSLOC)^{0.26}$ |
| Pages of Documentation | $= 34.7 * (KSLOC)^{0.93}$ |
| Annual Maintenance Cost | = 0.12 * (Development Cost) |
| Average Staff Size | $= 0.24 * (Effort)^{0.73}$ |

A new project could use this model and an estimate of KSLOC to estimate the other parameters. For example, if the estimated KSLOC were 100, then estimated

| | |
|---|---|
| Effort | = 135 staff-months |
| Duration | = 15 months |
| Pages of Documentation | = 2,514 |
| Average Staff Size | = 8-9 persons |

All of the above data were collected on NASA projects of similar characteristic. The groups within NASA doing similar work, under similar conditions can use these numbers reliably. Others will need to collect their own data to estimate their effort, size, etc.

Continued from page 4

## Why Isn't Cleanroom the Universal Software Development Methodology? (Part Two)

production line, and flaws are corrected if the quality is not as expected.

For software, this notion has been evolved so that you perform statistical usage testing—testing the software the way the users intend to use it. This is accomplished by defining usage probability distributions that identify usage patterns and scenarios with their probability of occurrence. Tests are derived that are generated based on the usage probability distributions. System reliability is predicted based on analysis of the test results using a formal reliability model, such as mean-time-to-failure [3].

The underlying concern is that random, statistical-based testing will not provide sufficient coverage to ensure a reliable product is delivered to the customer. The coverage concern stems from a misapprehension that statistical implies haphazard, large, and costly and that critical software requirements, which may be statistically insignificant, are overlooked or untested. Coverage is directly related to the robustness of the usage probability distributions that control the selection process and has not proven to be a problem in current applications of the methods. In a study performed by Dyer on the level of requirements coverage using statistical testing, 100 percent of the high-level requirements were covered, 90 percent of the subcomponent-level requirements were covered, and approximately 80 percent of all requirements were covered [3].

The Cleanroom method asserts that statistical usage testing is many times more efficient than traditional coverage testing in improving the reliability of software. Statistical testing, which tends to find errors in the same order as their seriousness (from a user's point of view), will uncover failures several times more effectively than by randomly finding errors without regard to their seriousness. The basis for software reliability starts with the definition of a statistical model, generally based on the concept that input data comes in at random times and with random contents. With defined initial conditions, any such fixed use is distinguishable from any other use. These uses can be assembled into a sequence of uses, and the collection identified as a stochastic process subject to evaluation using statistical methods.

Coverage testing is anecdotal and can only provide confidence about the specific paths tested. No assessment can be made about the paths not tested. Because usage testing exercises the software the way the users intend to use it, high-frequency errors tend to be found early. For this reason, statistical usage testing is more effective at improving software reliability than is coverage testing. Coverage testing is as likely to find a rare execution failure as it is to find a frequent one. If the goal of a testing program is to maximize the expected mean-time-to-failure, hence the reliability of the system, a strategy that concentrates on failures that occur more frequently is more effective than one that has an equal probability of finding high- and low-frequency failures [6].

Human functional verification has proven to be surprisingly synergistic with statistical testing according to Mills, Dyer, and Linger [4]. Experimental data from projects where both Cleanroom verification and more traditional debugging techniques were used show that the Cleanroom-verified software exhibited fewer errors injected. Those errors were less severe (possibly attributable to the philosophy of design simplification) and required less time to fix [1,2].

For more information about the author, please refer to part one of this article in the July - Aug. issue of this newsletter.

*References on page 9*

# ISO Registration for Software and System Providers - A Trip Report (or More Than You'll Ever Want to Know About ISO 9001)

*Carolyn Owens*

"Do what you say, say what you do, and document, document, document!" is apparently what many people think the ISO 9000 series standards require. According to the instructors of the three day U.C. Berkeley Extension class on ISO Registration for Software and System Providers, this thinking is incorrect. An organization's business practices must comply with the ISO standard and documentation should sufficiently and effectively support these business practices.

### An Overview of ISO 9000

Contrary to the belief that ISO is an acronym, it is the name of the International Organization for Standardization, a world wide federation of national standards bodies and comes from the Greek word "isos" meaning equal. ISO 9000 is a series of standards dealing with quality systems, with ISO 9001 being the most comprehensive. That is:

- ISO 9001 covers Product, Design, Development, Production, Installation and Servicing
- ISO 9002 covers Production, Installation and Servicing
- ISO 9003 covers Final Inspection and Test

Since ISO 9001 covers product design and development, it is the standard that is applied to software. As the process of developing and maintaining software is significantly different from other types of products, ISO produced the guidance document ISO 9000-3, (or ISO 9000 part 3) *Guidelines for the application of ISO 9001 to the development, supply and maintenance of software*. However, a problem exists in that the ISO 9001 standard was originally written in 1987 and revised in 1994 and the ISO 9000-3 was written in 1991 referencing the ISO 9001:1987. So there are some discrepancies. To get around these discrepancies, the class instructors advised us to become familiar with ISO 9000-3 but to work from ISO 9001:1994, as our organizations would be audited to this standard in seeking ISO 9001 registration.

According to the class instructors, the objectives of ISO 9000 are to specify a single set of quality system requirements for "all products, all processes, all industries, and all countries". They also pointed out that these standards are "common sense and good business practices". The primary reason why so many organizations are seeking ISO registration is the market place is requiring it, particularly in Europe. Other reasons include gaining a competitive edge and improving the organization's internal quality.

### An Overview of ISO 9000 Registration

ISO 9000 registration involves an organization hiring an accredited registrar to perform an ISO 9000 registration audit. (Note: Each country may have its own accreditation board and registrars accredited by that board. So an American firm may use a registrar accredited in the U.S. or one accredited by another country. The choice is usually market driven, that is, where does the firm plan to sell its

*Continued on page 10*

*References to Why Isn't Cleanroom the Universal Software Development Methodology?*

1. *Linger, R.C., "Cleanroom Process Model," IEEE Software, March 1994., pp. 50-58.*
2. *Hausler, P.A., R.C. Linger, and C.J. Trammel, "Adopting Cleanroom Software Engineering with a Phased Approach," IBM Systems Journal, Vol. 33, No.1, 1994, pp. 89-109.*
3. *Dyer, M., The Cleanroom Approach to Quality Software Development, John Wiley & Sons, Inc., New York (1992).*
4. *Mills, H.D., M. Dyer, and R.C. Linger, "Cleanroom Software Engineering," IEEE Software, September 1987, pp. 19-24.*
5. *Dyer, M., and A. Kouchakdjian, "Correctness Verification: Alternative to Structural Software Testing," Information and Software Technology, January/February 1990, pp. 53-59.*
6. *Cobb, R.C., and H.D. Mills, "Engineering Software under Statistical Quality Control," IEEE Software, November 1990, pp. 44-54.*

*Continued from page 9*

## ISO Registration for Software and System Providers - A Trip Report (or More Than You'll Ever Want to Know About ISO 9001)

products? The registrar audits the organization's business systems to verify that they comply with ISO 9001 (or ISO 9002, or ISO 9003, depending on which ISO registration the organization is seeking) and that they are effective, i.e. implemented.

After receiving the initial ISO registration, the organization must be re-audited by the registrar every 6 months in order to maintain its ISO registration. The class instructors stated that "ISO registration is more difficult to maintain than it is to obtain".

### Preparing for ISO Registration

There are five steps to preparing for ISO registration. These steps should look very familiar to software developers. Here is a brief summary of each step.

#### 1. Define the Requirements.

Management must decide on the objectives and goals for the ISO registration process. This should include producing a project plan which addresses the schedule, budget, and resource issues.

#### 2. Design and Develop the Organization's Policies and Procedures.

The primary objective for documentation is to effectively communicate the organization's values, goals, objectives, plans, policies, and procedures to all members of the organization. An additional objective is to obtain a corporate memory (e.g. records, logs, etc.). This documentation should be effective, efficient, and dynamic. It should also be concise (Note: The class instructors recommend erring on the side of too little rather than too much.) and reviewed and iterated upon.

One technique to aid organizations in developing procedures is to use a technique called process mapping. This is a visual summary of a process that helps provide understanding of the process. It also identifies opportunities for process improvement. Techniques such as data flow diagrams, flow charts and deployment charts can be used for process mapping.

A key piece of documentation that must exist for ISO registration is the quality manual. ISO provides in depth guidance as to what should be included in the quality manual, but essentially it must include a summary of the organization's interpretation of ISO 9000; the organization's quality policy statement; the organization's policies and procedures including the mission statement with reference to specific goals, the organization's structure including major activities and responsibilities, corrective action procedures such as internal audits, customer feedback mechanisms, and reviews and evaluation procedures or pointers to other documentation containing the above information.

#### 3. Test and Review.

During this step the documentation produced during step 2 is tested in the actual work environment using members of the organization who have not been involved in the design and development phase. Testing ensures that the training and necessary support is adequate to achieve successful implementation of the procedures.

#### 4. Implementation

This is the actual roll out of the policies and procedures to the whole organization. Throughout this initial roll out there should be observation, monitoring and support provided to update and improve the documentation.

#### 5. Implementation Review.

This is the final wrap-up meeting when lessons learned and opportunities for improvement can be documented.

The final emphasis of the class was on management's commitment to ISO implementation and the selection of the implementation project manager. The success of any project depends on management support, good project leadership and project planning. These are also the keys to success for a software organization to achieve ISO 9000 certification.

# Local Lab experts offer advice, expertise

**Reviews and Walkthroughs**
Carmen Parrish
Warren Persons, 2-3349
Jeff Young
Carolyn Owens

**Performance, Reliability & Safety**
Dennis Lawrence, 3-7828

**Reverse Engineering**
Jeff Young
Al Leibee

**Requirements Modeling/OOD**
Debbie Sparkman, 2-1855

**Testing**
Warren Persons, 2-3349
Nancy Storch, 2-8942
Al Leibee

**Software Quality Assurance**
Warren Persons, 2-3349

**CASE Tools**
Suzanne Pawlowski
Jeff Young

**Configuration Management**
Al Leibee
Carmen Parrish

**Project Estimation/Management**
Howard Guyer
Carolyn Owens

**JAD/FIND**
Candy Wolfe

If you need consulting help with a project involving software engineering, consider contacting one of the local LLNL experts.

## Software Engineering Working Group:

Bill Aimonetti, 3-2678
Bill Buckley, 3-4581,
Bob Corey, 3-3271
Antonia Garcia, 3-9884
Howard Guyer, 3-7671
Al Leibee, 2-1665
Judith Littleton, 3-4403
Donna Nowell, 2-1515
Jerry Owens, 2-1646
Carolyn Owens, 3-6085
Carmen Parrish, 2-9810
Suzanne Pawlowski, 3-0115
Frank Ploof, 2-6990
Terri Quinn, 3-2385
Denise Sumikawa, 2-1831
John Tannahill, 3-3514
Booker Thomas, 3-8800
Ernie Vosti, 3-0604
Jeff Young, 3-8333
Bill Warren, 2-5331
Candy Wolfe, 2-1863

The SEWG meetings are normally held the 1st and 3rd Thursday of each month at 3:00 p.m., in Bldg. 218, R114. The next meeting will be Thursday October 19.

# Upcoming Seminars and Conferences

**October**

30-3 C++ World Users Conference and Exhibition
Fairmont Hotel
Chicago, IL
To register call (212) 242-7515
WWW: http://www.sigs.com/

**November**

6-8 Current Success & Future Directions
The 4th SEI Conference on Software Risk
Monterey, CA
To Register call (412) 268-5800
WWW: http://www.sei.cmu.edu

29-30 Twentieth Annual
Software Engineering Workshop
NASA Goddard Space Flight Center
Greenbelt, Maryland
To register call (301) 286-6347
WWW: http://fdd.gsfc.nasa.gov/seltext.html

For current information about Software Engineering Institute Events: WWW: http://www.sei.cmu.edu/SEI/events/SEI_cal_events.html


**Software Engineering Newsletter Staff**

**Technical Editors:**
Jeff Young,
(510) 423-8333, L-548, jeffyoung@llnl.gov

Al Leibee,
(510) 422-1665, L-307, leibee1@llnl.gov

**Newsletter Compositors & Designers:**
Jennifer Gibson,
(510) 423-8543, L-307, jlgibson@llnl.gov

**General Information or article submission:**
(510) 423-8543, stc@llnl.gov



*NOTICE*

*This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, Lawrence Livermore National Laboratory, nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.*

*Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising, product endorsement or commercial purposes. This work was performed under the auspices of the U.S. Dept. of Energy at LLNL under contract no. W-7405-Eng-48.*



***UCRL-AR-121011-95-9/10***


**To subscribe or unsubscribe to this newsletter : (510) 423-8543, stc@llnl.gov**